Análisis de performance en Bases de Datos NoSQL y Bases de Datos Relacionales

Luciano Marrero[1], Verena Olsowy [1], Fernando Tesone[1], Pablo Thomas[1], Lisandro Delia[1], Patricia Pesado[1]

[1] Instituto de Investigación en Informática LIDI
Facultad de Informática - Universidad Nacional de La Plata – Argentina
Centro Asociado Comisión de Investigaciones Científicas de la Provincia de Buenos Aires

{lmarrero, volsowy, ftesone, pthomas, ldelia, ppesado}@lidi.info.unlp.edu.ar

Resumen: Los Sistemas de Gestión de Bases de Datos (SGBDs) no relacionales (NoSQL) surgen como una alternativa de solución a problemas no resueltos eficientemente por los SGBDs tradicionales. NoSQL, a diferencia del modelo relacional, no responde a un tipo de Base de Datos, sino que representa un conjunto de tipos de Bases de Datos, con diferentes formas y características para representar la información. Este trabajo representa la continuación del estudio presentado en CACIC 2019 [1] y tiene como objetivo comparar y analizar cuatro motores de Bases de Datos NoSQL y un motor de Base de Datos relacional utilizando diferentes esquemas bajo un gran volumen de datos.

Keywords: Bases de Datos Relacionales, Bases de Datos NoSQL, Almacenamiento Clave-Valor, Almacenamiento Documental, Almacenamiento de Familia de Columnas, Almacenamiento Orientado a Grafos.

1 Introducción

Actualmente, la mayoría de las aplicaciones son multiplataforma. El avance en la comunicación digital, el acceso constante a la información y el aumento en la utilización de tecnología móvil genera que estas aplicaciones sean de uso común por millones de usuarios simultáneamente. El modelo relacional de Codd (1970), que dio origen a los sistemas de Bases de Datos Relacionales, es sin duda, el modelo predominante de almacenamiento de información. Sin embargo, la idea de considerar que un único modelo de datos pueda adaptarse de forma eficiente a todos los requerimientos, ha sido discutida, y ha dado lugar a un conjunto de alternativas que plantean almacenar la información de forma no estructurada. Surgen así, otros motores de Bases de Datos que poseen implementaciones propias no relacionales y se denominan Bases de Datos NoSQL (No solo SQL). Estas Bases de Datos son adecuadas por su escalabilidad y tienden a utilizar modelos de consistencia relajados con el objetivo de lograr mayor performance y disponibilidad [2, 3, 4, 12]. Existe una gran variedad de motores de Bases de Datos NoSQL que se pueden catalogar en una de estas cuatro categorías:

Almacenamiento Clave/Valor: simples en cuanto a su implementación, almacenan datos como un conjunto de pares "clave/valor" (key-value). La clave (key) representa un identificador único que puede retornar un objeto denominado valor (value). Por ejemplo, Redis y Amazon DynamoDB, entre otros, implementan este tipo de almacenamiento [20, 21].

Almacenamiento Documental: el concepto central de este tipo de almacenamiento es el documento. Una Base de Datos NoSQL Documental, almacena, recupera y gestiona datos de documentos. Estos documentos encapsulan y codifican datos o información bajo algún formato estándar (XML, YAML, JSON, BSON). Por ejemplo, MongoDB y Apache CouchDB, entre otros, son implementaciones de Bases de Datos Documentales [5, 8, 17, 22].

Almacenamiento de Familia de Columnas: en este tipo de almacenamiento los datos se encuentran organizados por columnas, en lugar de por filas. En estas Bases de Datos Columnares por cada entrada, hay una columna, por lo tanto, los datos de cada entrada están dispuestos uno debajo del otro (y no uno al lado del otro, como en la variante orientada a filas). El objetivo de este tipo de almacenamiento es acceder con mayor precisión a la información que se necesita para responder a una consulta. Además, elimina la necesidad de explorar y descartar datos no deseados sobre una fila. Por ejemplo, Apache Cassandra y Apache HBase, entre otros, utilizan este tipo de almacenamiento [8, 9, 19, 23].

Almacenamiento de Grafos: en este tipo de almacenamiento se representa a la Base de Datos bajo el concepto de un grafo. Permite almacenar la información como nodos de un grafo y sus respectivas relaciones (aristas) con otros nodos. Aplica la teoría de grafos y son muy útiles para almacenar información en modelos que poseen numerosas relaciones entre sus datos. Por ejemplo, Neo4j y OrientDB, entre otros, permiten este tipo de almacenamiento [11, 24].

Así como las propiedades de ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad) están presentes en las Bases de Datos Relacionales, en las Bases de Datos NoSQL se tienen las propiedades BASE (**B**ase **A**vailability, **S**oft State, **E**ventual Consistency). Estas propiedades tienen su pilar principal en la disponibilidad. El sistema estará básicamente disponible todo el tiempo (**BA**), no necesariamente será consistente (**S**), pero en algún momento lo será, y eventualmente estará en un estado conocido (**E**). Las propiedades BASE difieren de las propiedades ACID, mientras ACID es más rígido y fuerza la consistencia, BASE permite un estado de consistencia flexible. Esto permite grandes niveles de escalabilidad [4, 7, 10, 13].

Este trabajo se centra en un análisis comparativo de rendimiento para cuatro casos de estudio, implementados en 5 motores de Bases de Datos distintos, 4 NoSQL (Cassandra, MongoDB, Neo4j y Redis) y un motor de Base de Datos Relacional (MySQL). Se plantean un conjunto de consultas para cada caso de estudio y para cada motor de Bases de Datos. Se realizaron 10 consultas para MySQL, 7 para MongoDB, 8 para Cassandra, 8 para Neo4j y 2 para Redis. En total se realizaron pruebas con 35 consultas utilizando un gran volumen de información. Cada caso de estudio posee sus propias características con el objetivo de comparar el desempeño de diferentes motores de Bases de Datos en una diversidad de escenarios.

A partir de la sección 2 el trabajo se organiza del siguiente modo: se explican características generales de los motores de Bases de Datos seleccionados, en la sección 3 se presentan los casos de estudio, en la sección 4 se analizan los resultados obtenidos y finalmente se presentan las conclusiones y trabajos a futuro.

2 Motores de Bases de Datos utilizados

Para realizar la experimentación presentada en este trabajo, se seleccionaron los siguientes motores de Bases de Datos: MySQL (Almacenamiento Relacional), MongoDB (Almacenamiento Documental), Apache Cassandra (Almacenamiento de Familia de Columnas), Neo4j (Almacenamiento Orientado a Grafos) y Redis (Almacenamiento Clave-Valor). Los motores de Bases de Datos NoSQL elegidos son los más populares en sus categorías [14].

MySQL es uno de los sistemas de Base de Datos Relacional más reconocido y popular del mercado con licencia dual (licencia pública general y licencia comercial). Es utilizado por numerosas empresas a nivel mundial, entre ellas se destacan, Facebook, Twitter, YouTube, GitHub, Booking.com, Spotify [15, 16].

MongoDB es la Base de Datos NoSQL Documental con mayor popularidad, es de código abierto y multiplataforma. Diferentes empresas como Telefónica, Facebook, Nokia, entre otras, hacen uso de MongoDB [5, 6, 17, 18].

Apache Cassandra es la Base de Datos NoSQL con almacenamiento en Familia de Columnas con mayor popularidad. Es multiplataforma, diseñada para trabajar de forma distribuida con grandes volúmenes de datos. Empresas como Instagram, Netflix, GitHub, entre otras, hacen uso de Cassandra [19, 21].

Neo4j es la Base de Datos Orientada a Grafos (BDOG) con mayor popularidad, escrita en JAVA y posee licencia dual (comercial y AGPL). Se basa en Grafos para representar datos y las relaciones entre ellos. Empresas como Walmart, Ebay y Cisco hacen uso de Neo4j [11, 24].

Finalmente, *Redis* es la Base de Datos Clave-Valor con mayor popularidad. Redis es una Base de Datos que opera sus estructuras de datos en memoria primaria, ofrece alto rendimiento, posee persistencia opcional y tiene licencia de Software Libre BSD. Es utilizada por diferentes empresas, Stack Overflow, Twitter, GitHub, entre otras [20].

3 Casos de estudio

Para cada caso de estudio se realizó previamente el Modelo Conceptual de Datos utilizando un Diagrama de Entidad Relación (DER). Posteriormente, se realizó la derivación al esquema físico correspondiente a cada motor de base de datos utilizado.

En MySQL se generó el modelo relacional. En MongoDB las entidades se representan mediante colecciones de documentos BJSON y las relaciones se expresan mediante documentos embebidos [1, 2, 17]. En Cassandra, los datos no se almacenan

en términos de entidades, sino que se representan en términos de consultas, por lo tanto, es importante definir las consultas a realizar [1, 2, 9, 19]. En el caso de Neo4j, los datos se representan mediante nodos y las relaciones son las aristas entre éstos [2, 24]. En el caso de Redis, al ser una Base de Datos en memoria principal, se pensó en el conjunto de claves que serían necesarias tener vigentes de acuerdo con las consultas a realizar [2, 20].

La ejecución de todas las pruebas se realizó utilizando la misma memoria y sistema operativo (4GB de RAM, Ubuntu versión 18.04). En MySQL se utilizó la versión 5.7, en MongoDB la versión 4.0, en Cassandra la versión 3.11, en Neo4j la versión 4.1 y en Redis la versión 4.0. Para cada caso de estudio el volumen de datos utilizado se generó de forma aleatoria.

En los 3 primeros casos de estudios no fue posible evaluar Redis debido a que el gran volumen de datos utilizado no se puede gestionar adecuadamente en memoria principal.

A continuación, se presentan los 4 casos de estudio. A modo de ejemplo y para no exceder la cantidad de páginas, sólo en el primero de ellos se presentan los esquemas físicos y el código de las consultas. No obstante, en todos los casos se expresa el DER y los resultados obtenidos.

3.1 Caso de estudio 1

Se presenta un esquema para la bandeja de entrada de un sistema de mensajería interna. Se han generado de forma aleatoria 30.000.000 de mensajes en total correspondientes a 6.000.000 de usuarios distintos. La Figura 1 presenta el DER del problema en cuestión.

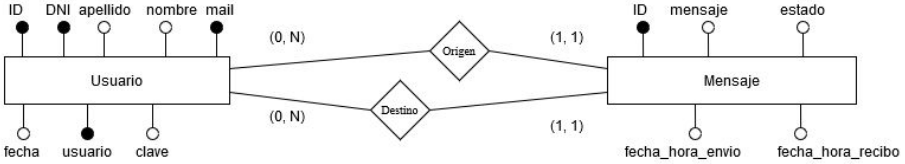

Figura 1. Esquema Conceptual expresado mediante un DER.

La Figura 2 presenta el modelo relacional (MySQL) resultante de la Figura 1.

```
Usuario=(id, DNI, apellido, nombre, mail, fecha_alta, usuario, clave)
Menaje=(id, mensaje, estado, fecha_hora_env, fecha_hora_rec,
usuario_orig(FK), usuario_dest(FK))
```

Figura 2. Esquema Relacional.

La Figura 3 muestra la estructura del documento para MongoDB.

```
Mensaje = { _id, mensaje, estado, fecha_hora_env, fecha_hora_rec,
   usuario_origen={'dni','nombre','apellido', 'mail','usuario'},
   usuario_destino={'dni','nombre','apellido', 'mail','usuario'}}
```

Figura 3. Documento para MongoDB.

En Cassandra, el esquema físico se expresa de acuerdo a las consultas a realizar. La Figura 4 plantea dicho esquema físico.

```
Mensajes_por_usuario =((usuario_orig, usuario_dest,
fecha_hora_env)PK, mensaje, fecha_hora_rec, estado);
```

Figura 4. Esquema Físico de Cassandra.

La Figura 5, muestra una parte del almacenamiento de Grafos planteado para Neo4j. Los nodos de color verde representan a los usuarios, mientras que los grises representan mensajes.

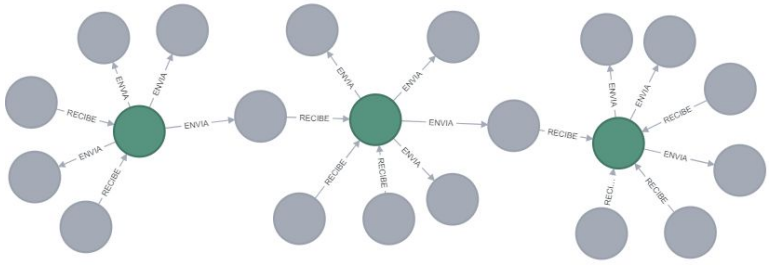

Figura 5. Parte del Grafo generado para el caso de estudio 1.

A continuación, se plantean 3 consultas para evaluar el rendimiento en este caso de estudio.

"C1: Mensajes entre dos usuarios específicos"

"C2: Buscar una cadena de caracteres en los mensajes de un usuario específico"

"C3: Mensajes recibidos y no leídos por un usuario específico"

Las Figuras 6, 7, 8 y 9 muestran la resolución de C1 para 4 motores de Base de Datos (MySQL, Cassandra, MongoDB y Neo4j).

```
SELECT mensaje, fecha_hora_env, estado
FROM mensaje m INNER JOIN usuario o ON (o.id=m.usuario_orig) INNER
JOIN usuario d ON (d.id=m.usuario_dest) WHERE (o.usuario =
'user256' AND d.usuario = 'user1') OR (o.usuario = 'user1' AND
d.usuario = 'user256') ORDER BY m.fecha_hora_env DESC
```

Figura 6. Implementación de C1 en MySQL.

```
SELECT mensaje, fecha_hora_env, estado
FROM mensajes_por_usuario
WHERE usuario_orig IN ('user256','user1') AND usuario_dest IN
('user1', 'user256') ORDER BY fecha_hora_env DESC
```

Figura 7. Implementación de C1 en Cassandra.

```
db.mensajes.find({$or:[ {$and:[{"usuario_origen.usuario": "user1"},
              {"usuario_destino.usuario": "user259"}]},
       {$and:[{"usuario_origen.usuario": "user259"},
               {"usuario_destino.usuario": "user1"}]}
]},{mensaje:1, fecha_hora_env:1, estado:1}).sort({fecha_hora_env: -1})
```

Figura 8. Implementación de C1 en MongoDB.

```
MATCH (u1:Usuario)-[r1]-(m:Mensaje)-[r2]-(u2:Usuario)
WHERE ID(u1)=34000550 AND ID(u2)=34000805
RETURN u1.usuario, r1, m, r2, u2.usuario;
```

Figura 9. Implementación de C1 en Neo4j.

En la Tabla 1 se expresan los tiempos obtenidos en segundos de la ejecución de cada consulta.

Tabla 1. Resultados del caso de estudio 1 expresados en segundos.

Consultas	MySQL	MongoDB	Cassandra	Neo4j
C1	0.33	385	0.35	0.24
C2	259	178	9	0.15
C3	20	166	72	0.24

3.2 Caso de estudio 2

Este caso de estudio representa una central de monitoreo que registra y procesa eventos procedentes de distintas fuentes. Se han generado aleatoriamente 40.000.000 de eventos en total. La Figura 10 presenta el DER del problema descripto.

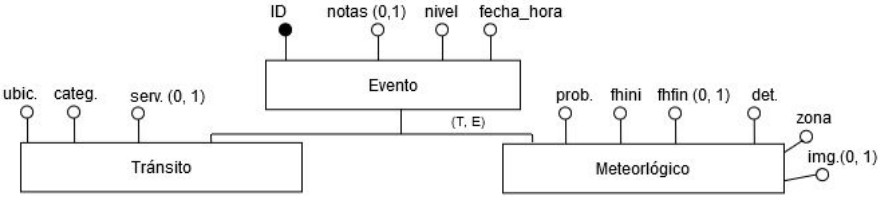

Figura 10. Esquema Conceptual expresado mediante un DER.

A continuación, se plantean 3 consultas para evaluar el rendimiento en este caso de estudio.

"C1: Eventos meteorológicos que superen una determinada probabilidad de ocurrencia"

"C2: Eventos de tránsito con una determinada categoría en un período de tiempo"

"C3: Cantidad de eventos de un determinado nivel de prioridad (Alto, Medio o Bajo) "

La Tabla 2 expresa los tiempos obtenidos en segundos de la ejecución de cada consulta.

Tabla 2. Resultados de este caso de estudio expresado en segundos

Consultas	MySQL	MongoDB	Cassandra	Neo4j
C1	200	119	123	360
C2	233	117	120	134
C3	81	106	200	288

3.3 Caso de estudio 3

En este caso, se plantea un esquema de vuelos pertenecientes a distintas aerolíneas. Se han generado aleatoriamente la información de 40.000.000 de vuelos entre 10.000 aeropuertos distintos. La Figura 11 presenta el DER del problema en cuestión.

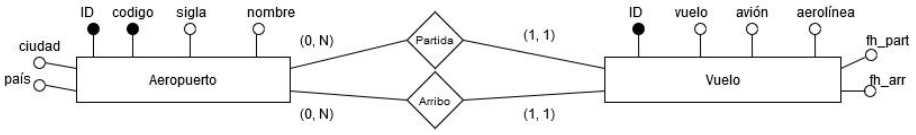

Figura 11. Esquema Conceptual expresado mediante un DER.

A continuación, se plantean 2 consultas para evaluar el rendimiento en este caso de estudio.

"C1: Vuelos directos entre un origen y un destino determinado"

"C2: Vuelos que posean una escala entre un origen y un destino determinado"

La Tabla 3 expresa los tiempos obtenidos en segundos de la ejecución de cada consulta.

Tabla 3. Resultados del caso de estudio 3 expresados en segundos

Consultas	MySQL	MongoDB	Cassandra	Neo4j
C1	18	166.	1	0,1
C2	3720	ver sección 4	1.5	0.4

3.4 Caso de estudio 4

El cuarto y último caso de estudio presenta un esquema elegido para evaluar específicamente Redis en comparación con MySQL. Se representa una caché de

búsquedas recientes. Redis, si bien brinda la posibilidad de persistencia, se destaca por disponer sus estructuras de datos en memoria principal. Para este caso de estudio se han generado 100.000 usuarios y 10.000.000 búsquedas en total. La Figura 12 presenta el DER del problema en cuestión. La entidad "Búsqueda" posee un identificador mixto, el cual se compone del identificador del usuario más la fecha y hora de la búsqueda.

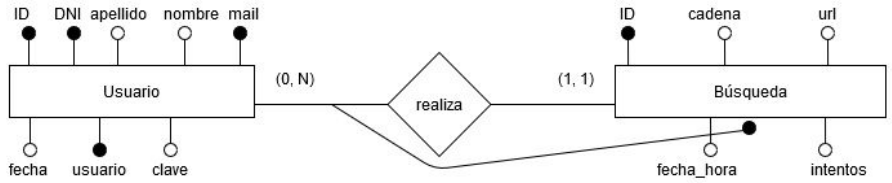

Figura 12. Esquema Conceptual expresado mediante un DER.

A continuación, se plantean 2 consultas para evaluar el rendimiento en este caso de estudio

"C1: Las 5 búsquedas más recientes para un usuario determinado"
"C2: Las 5 búsquedas más frecuentes para un usuario determinado."

La Tabla 4 expresa los resultados obtenidos en microsegundos.

Tabla 4. Resultados del caso de estudio 4 expresado en microsegundos.

Consultas	MySQL	Redis
C1	580000	0,48
C2	550000	0,53

4 Análisis de resultados

4.1 Análisis de caso de estudio 1

En este primer caso de estudio, Neo4j obtiene el mejor rendimiento. No obstante, debido a que la cantidad de nodos "mensajes" tiene crecimiento exponencial, el rendimiento puede degradarse.

Para la consulta 1, Cassandra y MySQL obtienen tiempos próximos a Neo4j. Esto se debe a la característica de los campos utilizados en el filtro de la consulta.

MongoDB obtiene tiempos superiores a los otros 3 motores de Bases de Datos. Las consultas que involucran campos que pertenecen a documentos embebidos pueden afectar su performance.

Para la consulta 2, Cassandra obtiene mejor tiempo que MySQL y MongoDB pero necesita de la implementación de un índice para realizar búsquedas relativas. MongoDB obtiene mejor rendimiento que MySQL; las búsquedas relativas y el uso de JOINs afectan el rendimiento en MySQL.

Para la consulta 3, MySQL obtiene mejor rendimiento que MongoDB y Cassandra. Realizar consultas que involucren columnas que no son parte de la Primary Key o Clustering Key en Cassandra, afecta su rendimiento.

4.2 Análisis de caso de estudio 2

En este caso de estudio, ninguno de los 4 motores de Bases de Datos obtiene el mejor rendimiento para todas las consultas planteadas.

En las consultas 1 y 2, MongoDB obtiene el mejor rendimiento. En estas consultas, el esquema planteado para MongoDB no utiliza documentos embebidos y no requiere almacenar campos opcionales, evitando comparaciones nulas. En MySQL las operaciones de JOINs afectan su performance. En Cassandra, las consultas planteadas involucran campos que no están presentes en la Primary Key o Clustering Key, lo que afecta su tiempo de respuesta. En Neo4j el esquema del caso de estudio no es apropiado para el Almacenamiento de Grafos, dado que no existen relaciones entre los nodos del grafo.

En la consulta 3, MySQL optimiza la función de agregación COUNT, mejorando así, su tiempo de respuesta en comparación a los otros 3 motores de Bases de Datos.

4.3 Análisis de caso de estudio 3

En este caso de estudio Neo4j obtiene el mejor rendimiento para las consultas planteadas. Esto es debido a que dichas consultas son resueltas eficientemente en el Almacenamiento de Grafos.

Cassandra, obtiene mejores tiempos que MySQL y MongoDB, esto se debe a que el esquema físico en Cassandra se plantea en términos de consultas. En MySQL la baja de performance entre la consulta 1 y la consulta 2 se debe a los JOINs necesarios para resolver cada consulta.

4.4 Análisis de caso de estudio 4

En este caso de estudio se plantean dos consultas para Redis y MySQL. Redis obtiene el mejor rendimiento. Esto se debe a que Redis define íntegramente sus estructuras en memoria RAM donde opera los datos, mientras que MySQL por su parte distribuye su procesamiento entre memoria RAM y memoria secundaria.

5 Conclusiones

Este trabajo se concentra en un estudio comparativo entre 5 motores de Bases Datos (4 NoSQL y 1 Relacional) en 4 casos de estudios diferentes. En primer lugar, se

realizó una breve descripción de las diferentes formas de almacenamiento de los motores de Bases de Datos NoSQL, junto a sus propiedades específicas. Los 4 casos de estudios planteados presentan diferentes esquemas de información y para cada uno de ellos se definió un conjunto de consultas específico, resultando 35 consultas en total. Para el primero de ellos se presenta el desarrollo en forma completa, es decir, el esquema DER, el esquema físico y el código de cada una de las consultas. En los 3 casos restantes se presentó el DER.

Los resultados obtenidos en cada uno de los 4 casos de estudio se han analizado con el criterio de justificar porqué en cada caso un motor se comporta mejor que otro.

Para finalizar, se puede concluir que si se dispone un gran volumen de información no hay un motor de Bases de Datos que obtenga el mejor rendimiento para satisfacer todas las consultas planteadas.

En la medida que aumente el volumen de datos, es necesario evaluar qué alternativa de almacenamiento, es decir, que tipo de motor de Bases de Datos, es conveniente utilizar con el objetivo de brindar mejores prestaciones.

Ante la variedad de resultados obtenidos, y la falta de prevalencia de un motor de Bases de Datos con el mejor rendimiento, debe realizarse escalamiento horizontal (aumentar nodos) y plantear una nueva evaluación en ese nuevo contexto.

6 Trabajo Futuro

Como trabajo futuro, se prevé incorporar nuevas pruebas y reforzar las existentes con alternativas de motores de Bases de Datos Relacionales y No Relacionales. Además, se buscará explorar la capacidad de escalamiento horizontal de las Bases de Datos NoSQL y de las Bases de Datos Relacionales. Finalmente, se prevé experimentar con Bases de Datos NewSQL y Bases de Datos de Series Temporales.

7 Bibliografía

1. Un estudio comparativo de bases de datos relacionales y bases de datos NoSQL. Pesado Patricia Mabel, Thomas Pablo, Delía Lisandro, Marrero Luciano, Olsowy Verena, Tesone Fernando, Fernandez Juan Sosa. XXV Congreso Argentino de Ciencias de la Computación (CACIC 2019). Universidad Nacional de Río Cuarto, Córdoba, 14 al 18 de octubre de 2019. ISBN 978-987-688-377-1. http://sedici.unlp.edu.ar/handle/10915/91403.
2. Aspectos de la Ingeniería de Software, Bases de Datos Relacionales y Bases de Datos No Relacionales para el desarrollo de Sistemas de Software en Escenarios Híbridos. XXII Workshop de Investigadores en Ciencias de la Computación (WICC 2020, Universidad Nacional de la Patagonia Austral, El Calafate, Santa Cruz, Argentina).
3. Aspectos de ingeniería de software y bases de datos para el desarrollo de sistemas de software en escenarios híbridos. XXI Workshop de Investigadores en Ciencias de la Computación (WICC 2019, Universidad Nacional de San Juan, San Juan, Argentina). ISBN 978-987-3619-27-4. http://sedici.unlp.edu.ar/handle/10915/77088.

4. Ajit Singh, Sultan Ahmad. Data Modeling with NoSQL Database. ISBN 978-1072978374 (2019).
5. Jitender Kumar, Varsha Garg. Security analysis of unstructured data in NOSQL MongoDB database. International Conference on Computing and Communication Technologies for Smart Nation (IC3TSN) 2017. https://ieeexplore.ieee.org/document/8284495.
6. Benymol Jose, Sajimon Abraham. Exploring the merits of nosql: A study based on MongoDB. International Conference on Networks & Advances in Computational Technologies (NetACT) 2017. https://ieeexplore.ieee.org/document/8076778.
7. Replicación de bases de datos NoSQL en dispositivos móviles. http://sedici.unlp.edu.ar/handle/10915/48085.
8. Roshni Bajpayee, Sonali Priya Sinha y Vinod Kumar. Big Data: A Brief investigation on NoSQL Databases. International Journal of Innovations & Advancement in Computer Science (IJIACS). ISSN 2347. Volume 4, Issue 1. January 2015.
9. Deepak Vohra. NoSQL Web Development with Apache Cassandra. Cengage Learning, 2015. ISBN 9781305576773.
10. NoSQL: modelos de datos y sistemas de gestión de bases de datos. XX Workshop de Investigadores en Ciencias de la Computación (WICC 2018, Universidad Nacional del Nordeste). http://sedici.unlp.edu.ar/handle/10915/67258
11. A Performance Optimization Scheme for Migrating Hive Data to Neo4j Database. Publicado en 2018 International Symposium on Computer, Consumer and Control (IS3C). ISBN: 978-1-5386-7036-1. https://ieeexplore.ieee.org/document/8644938
12. Carlo Batini, Stefano Ceri, Shamkant B. Navathe. Diseño Conceptual de Bases de Datos, un enfoque de entidades-interrelaciones. Addison-Wesley / Díaz de Santos. ISBN 0-201-60120-6 (1994).
13. Flexibilidad en bases de datos NoSQL sobre ambientes web mining. http://sedici.unlp.edu.ar/handle/10915/59099
14. Ranking de bases de datos según su popularidad. https://db-engines.com/en/ranking. Accedido en julio 2020.
15. MySQL. https://www.mysql.com/. Accedido en julio 2020.
16. Clientes de MySQL. https://www.mysql.com/customers/. Accedido en julio 2020.
17. MongoDB. https://www.mongodb.com/. Accedido en julio 2020.
18. Empresas que utilizan MongoDB. https://www.mongodb.com/who-uses-mongodb Accedido en julio 2020.
19. Apache Cassandra. http://cassandra.apache.org/. Accedido en agosto 2020.
20. Redis. https://redis.io/. Accedido en agosto 2020.
21. Amazon DynamoDB. https://aws.amazon.com/es/dynamodb/. Accedido en julio 2020.
22. Apache CouchDB. http://couchdb.apache.org/. Accedido en julio 2020.
23. Apache HBase. http://hbase.apache.org/. Accedido en julio 2020.
24. Neo4j. https://neo4j.com/. Accedido en julio 2020.